Perceptron Revisited

AW

Lecture Overview

1. Fitting a Model

2. Regularization

3. Neuron Variety

Overfitting vs Underfitting

- Fitting a model to training data set = ideally no errors (or small number of errors) on target variable in training data

Overfitting:

- Fitting model well ≠ good prediction performance on unseen data. If gap between training and testing performance is big model is *overfit*
- Often happens when model is complex and training set is small. We say that model fit to sample didn't *generalize* to data

Underfitting:

- Gap between performance on training data and testing data is acceptable (generalize well), but the number of errors is relatively big in in both cases. We say that model is *underfit*

Bias vs Variance – Intuition of Fitting a Model

- Let $X \times Y$ be data values (Y-target variable), and P is a distribution over $X \times Y$
- We assume that true data is related by function g and that there is added noise ε, i.e. $y = g(x) + \varepsilon$. Also noise has 0 mean and σ^2 variance
- Learning algorithm produces an *estimator* - a *function f* which tells us what value $\hat{y} \in \hat{Y}$ to expect in the future on data $x \in \mathrm{X}$.
- The estimator that we produce depends on the data D we used to build it, so $\hat{y} = f(x, D)$
- Whatever the algorithm that produces estimator we can look at the square of expected error of the algorithm

$$E_{P,\varepsilon}\left((y - \hat{y})^2\right) = E_{P,\varepsilon}\left((g(x) + \varepsilon - f(x, D))^2\right)$$

where P is the joint distribution on $X \times Y$, and the expectation ranges over different choices of the training set $D = \{(x_1, y_1), \ldots, (x_n, y_n)\}$, in which all data points are sampled from the same joint distribution $P(x, y)$

Bias vs Variance – Intuition of Fitting a Model

- Turns out that for any learning algorithm error of aproduced estimator can be written as

$$E_{P,\varepsilon}\left((y-\hat{y})^2\right) = BIAS\left(f(x,D)\right) + VAR\left(f(x,D)\right) + \sigma^2$$

- The *bias* of estimator $\hat{y} = f(x,D)$ relative to Y is defined as

$$BIAS\left(\hat{Y}, Y\right) := E_P[\hat{y} - y] = E_P[\hat{y}] - y$$

- Intuitively, the bias shows by how much the estimator misses the target (on average).
- *Variance*, of an estimator relative to Y is defined as

$$VAR\left(\hat{Y}\right) := E_P\left[\left(\hat{Y} - E_P[\hat{Y}]\right)^2\right] = E_P\left(\hat{Y}^2\right) - E_p\left(\hat{Y}\right)^2$$

- tells how wider or narrower are the differences between estimates are (i.e. how much the error of an estimator moves around its mean)

Thus the relationship between an estimator and the actual values we'll get can be described the *bias* and the *variance*.

- Overfitted data has low bias but high variance
- Underfitted data has high bias low variance

Example: Perceptron on Small Data Set

- Let data have 5 features $X_1, \ldots, X_5$. Suppose that actual function we are learning is $Y = 2X_1$, with other features being irrelevant and there is no noise.
- Let training dataset be

X_1	X_2	X_3	X_4	X_5	Y
1	1	0	0	0	2
2	0	1	0	0	4
3	0	0	1	0	6
4	0	0	0	1	8

- Perceptron with identity activation and SSE loss is may learn instead of $Y = 2X_1$ the function

$$Y = X_1 + X_2 + 2X_3 + 3X_4 + 4X_5$$

Which clearly is overfitting

Lecture Overview

1. Fitting a Model

2. Regularization

3. Neuron Variety

Regularization

- *Regularization is any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error.*
- Provably there is not best general form of regularization (No free lunch theorem – teach it in Data Mining class).
- Must be chosen with prior assumptions for a specific task at hand
- Can be viewed as a part of loss function. Then typically it is a penalty term for "unwanted" effects
- More often than not it is controlling complexity of the model
- Smaller absolute values of the parameters likely to overfit less. Penalizing parameter range is known as parameter norm penalties

Regularization Observations and Example

- Parameter norm penalty: to enforce smaller values we can add penalty $\lambda \|W\|^p$ to the loss function is used.
  - When $p = 2$ *it* is known as *Tikhonov regularization*.
  - When the norm is L^2 (Euclidean distance) the penalty is $\sum_i \lambda w_i^2$
- An example of a regularized version of perceptron learning on a training instance $(\vec{x}, y)$ and update step-size $\alpha > 0$ is:

$$\vec{w}' = \vec{w}(1 - \alpha\lambda) + \alpha e(\vec{x})\vec{x}$$

where, $e(x)$ represents the current error $(y - \hat{y})$ between observed and predicted values of training instance $\vec{x}$

  - This type of penalty is like a weight decay during the updates.

What is Regularization and When to Use It

- Regularization is particularly important when the amount of available data is limited.
- Possible interpretation of regularization is gradual forgetting, as a result of which "less important" (i.e., *noisy*) patterns are removed.
- In general, it is often advisable to use more complex models with regularization rather than simpler models without regularization.

Lecture Overview

1. Fitting a Model

2. Regularization

3. Neuron Variety

Deep Learning vs Traditional ML

- Exploring the neural models for traditional ML is useful because they are building blocks for deep learning.
- What does DL add to ML?
  - Add capacity with more nodes for more data.
  - Controlling the structure of the architecture provides a way to incorporate domain-specific insights (e.g., recurrent networks and convolutional networks).
  - In some cases, making minor changes to the architecture leads to interesting models, e.g. adding a sigmoid/softmax layer in the output of a neural model for (linear) matrix factorization can result in logistic/multinomial matrix factorization (e.g., word2vec) - later.

Linear Regression

- Predict a *real-valued* target variable Y given observed d-dimensional real-valued feature vector $(X_1, \ldots, X_d)$ assuming linear relationship $Y = w_1 X_1 + \cdots + w_n X_n$ where weights are unknown given training data that contains n different training pairs $(\vec{x}_1, y_1), \ldots, (\vec{x}_n, y_n)$, where each $\vec{\mathrm{x}}_i \in \mathbb{R}^d$ are data points, and each $y_i \in \mathbb{R}$ are values of a *real-valued* target variable
- Has well known closed form solution:

  Let $X = (\vec{x}_1 \quad \cdots \quad \vec{x}_n)^T$ so called design matrix, $\vec{w} = (w_1 \ldots w_n)^T$ unknown weights and $\vec{y} = (y_1 \ldots y_n)$ vector of values of target variables then $X\vec{w} = \vec{y}$ is the least square problem that the same solution as normal equation $X^T X\vec{w} = X^T\vec{y}$. If $X^T X$ is invertible then it unique solution $\vec{w} = (X^T X)^{-1} X^T \vec{y}$

Linear Regression Neuron

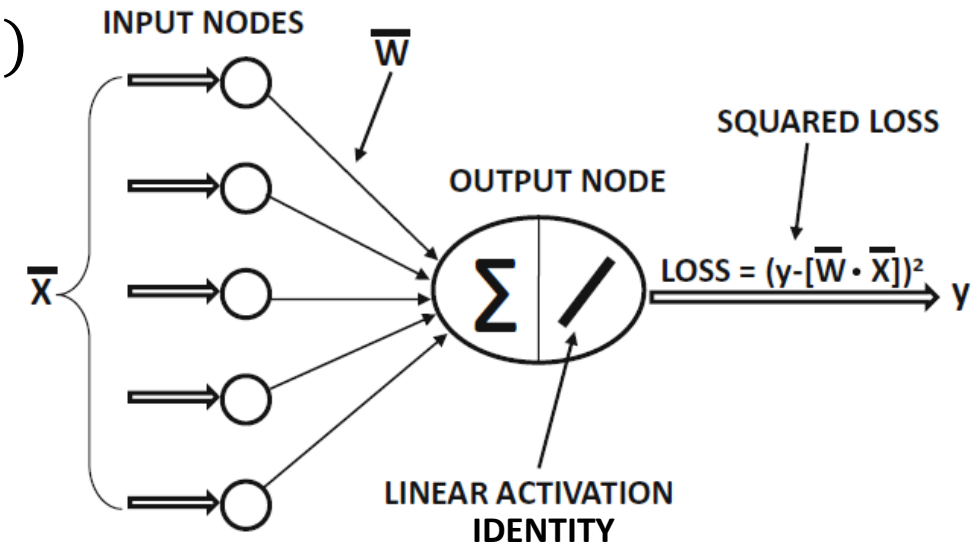
Linear regression problem can be solved by a perceptron with sum $z = \vec{x}_i \cdot \vec{w}$ followed by $\hat{y} = z$ – i.e. identity activation function and pointwise loss function $L_{\vec{x}} = \frac{1}{2}(y_{\vec{x}} - \hat{y}_{\vec{x}})^2$.

Given current value $\vec{w}_c$ of weights vector , for a data point (x_i, y_i) by chain rule

$$\begin{aligned}\nabla_{\vec{w}} L_{\vec{x}_i} &= \frac{\partial L_{\vec{x}_i}}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z} \cdot \nabla_{\vec{w}} z \\ &= \frac{1}{2}(-2)(y_i - \hat{y}_{\vec{x}_i}) \cdot \nabla_{\vec{w}} z \\ &= -\left(y_i - \hat{y}_{\vec{x}_i}\right) \cdot \nabla_{\vec{w}} (\vec{w} \cdot \vec{x}_i) \\ &= -\left(y_i - \hat{y}_{\vec{x}_i}\right)\vec{x}_i\end{aligned}$$

So for learning rate α the gradient updates are

$$\begin{aligned}\vec{w}_u &= \vec{w}_c + \alpha(-1)(-\left(y_i - \hat{y}_{\vec{x}_i}\right)\vec{x}_i) \\ &= \vec{w}_c + \alpha\left(y_i - \hat{y}_{\vec{x}_i}\right)\vec{x}_i\end{aligned}$$



Regression w/Regularization and Classification

- Regularization penalizing the loss function of least-squares regression for the magnitude of weight adds the term $\lambda\|\vec{w}\|^2$, where $\lambda > 0$ is the *regularization parameter*:

$$L_{\vec{x}} = \frac{1}{2}[(y_{\vec{x}} - \hat{y}_{\vec{x}})^2 - \lambda\vec{w} \cdot \vec{w}]$$

then $\nabla_{\vec{w}} L_{\vec{x}_i} = -[(y_i - \hat{y}_{\vec{x}_i})\vec{x}_i + \lambda\vec{w}_c]$ and update rule is

$$\vec{w}_u = \vec{w}_c(1 - \alpha\lambda) + \alpha\,(y_i - \hat{y}_{\vec{x}_i})\vec{x}_i$$

where w_c is the before update value of weights

- Note that the update rule without regularization is identical to the perceptron update but gives different results because in percepton $y_i, \hat{y}_i \in \{-1, +1\}$ since target variable is classification rather than regression, so the prediction is given with step-size activation (not 1).

Least-square Classification

- We can apply linear regression neuron directly for binary classification. It is known as *Widrow-Hoff learning.*
- The update (with or without regularization) is called *delta* rule
- Delta rule can be obtained directly by rewriting

$$L_{\vec{x}} = \frac{1}{2}[(y_{\vec{x}} - \hat{y}_{\vec{x}})^2 - \lambda\vec{w}\cdot\vec{w}]$$

$$= \frac{1}{2}[\underbrace{y_{\vec{x}}^2}_{1}(y_{\vec{x}} - \hat{y}_{\vec{x}})^2 - \lambda\vec{w}\cdot\vec{w}]$$

$$= \frac{1}{2}[(y_{\vec{x}}\cdot y_{\vec{x}} - y_{\vec{x}}\cdot\hat{y}_{\vec{x}})^2 - \lambda\vec{w}\cdot\vec{w}$$

$$= \frac{1}{2}[(1 - y_x\hat{y}_{\vec{x}})^2 - \lambda\vec{w}\cdot\vec{w}$$

So $\nabla_{\vec{w}}L_{\vec{x}_i} = -y_i\left[\left(1 - y_i\hat{y}_{\vec{x}_i}\right)\vec{x}_i + \lambda\vec{w}_c\right]$

The delta rule: $\vec{w}_u = \vec{w}_c(1 - \alpha\lambda) + \alpha y_i\left(1 - y_i\hat{y}_{\vec{x}_i}\right)\vec{x}_i$

Can be proven equivalent to Fisher Linear Discriminant for binary targets

Logistic Regression Problem

- Suppose we have features $x_1, \ldots, x_d \in \mathbb{R}$ so that each feature vector is real d- dimensional $\vec{x} \in \mathbb{R}^d$ and one Bernoulli target variable $y \in \{0,1\}$ distributed with probability $1-p, p$. Suppose also that there is joint probability on pairs $(\vec{x}, y)$ about which we know that there is a linear relationship between the feature variables and log-odds of the event that $y = 1$ i.e. $\ln \frac{p}{1-p} = w_1 x_1 + \cdots w_k x_k = \vec{w} \cdot \vec{x}$ for some unknown vector of weights $\vec{w}$. Then

  $$p\left(1 + e^{\vec{w}\cdot\vec{x}}\right) = e^{\vec{w}\cdot\vec{x}} \text{ and } p = \frac{e^{\vec{w}\cdot\vec{x}}}{1+e^{\vec{w}\cdot\vec{x}}} = \frac{1}{1+e^{-\vec{w}\cdot\vec{x}}}$$

- As before suppose we have a sample that contains n different training pairs $(\vec{x}_1, y_1), \ldots, (\vec{x}_n, y_n)$
- We are given an instance $\vec{x}$ and we want to predict associated target variable

Logistic Regression Neuron

- Our neuron will output $P(y|\vec{x})$ which means that to match training data for positive examples in the training data, we want to maximize $P(y_i = 1)$ and for negative examples, we want to maximize $P(y_i = -1)$. Equivalently for positive samples we want maximize $\hat{y}_i$ and for negative examples we want to maximize $1 - \hat{y}_i$. General expression for both cases is always maximize $\left|\frac{y_i}{2} - 0.5 + \hat{y}_i\right|$.
- The products of these probabilities must be maximized over all training instances to maximize the likelihood

$$L = \prod_{i=1}^{n} \left|\frac{y_i}{2} - 0.5 + \hat{y}_i\right|$$

It is maximized when its log is maximized which is easier (sums are easier than products) and instead of maximizing we minimize its negation, so we take as loss function

$-\ln L = -\ln\left(\prod_{i=1}^{n} \left|\frac{y_i}{2} - 0.5 + \hat{y}_i\right|\right) = \sum_{i=1}^{n} -\ln\left|\frac{y_i}{2} - 0.5 + \hat{y}_i\right|$ where

$L_{\mathrm{i}} = -\ln\left|\frac{y_i}{2} - 0.5 + \hat{y}_i\right|$ is loss per example that we need to minimize

Logistic Regression Neuron - Summary

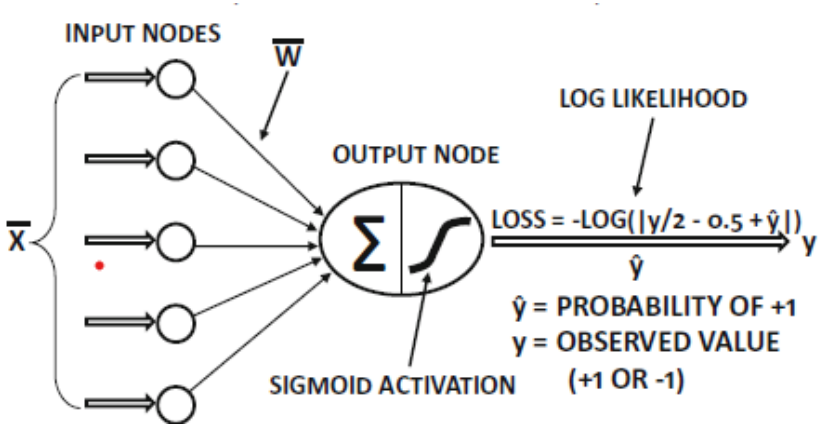
- So cumulative loss function ia

$$-\ln L = -\ln\left(\prod_{i=1}^{n}\left|\frac{y_i}{2} - 0.5 + \hat{y}_i\right|\right) = \sum_{i=1}^{n} -\ln\left|\frac{y_i}{2} - 0.5 + \hat{y}_i\right|$$

  - per example loss function

$$L_{\mathrm{i}} = -\ln\left|\frac{y_i}{2} - 0.5 + \hat{y}_i\right|$$

So neuron is:



Where output $\hat{y}$ is predicted probability of 1,

$z = \sum_i w_i x_i$ activation input

Logistic Regression Neuron

$L_{\mathrm{i}} = -\ln\left|\frac{y_i}{2} - 0.5 + \hat{y}_i\right|$ where $\hat{y}$ is output of activation part

with $\Phi(z) = \frac{1}{1+e^{-z}}$ that has input $z = \vec{w} \cdot \vec{x}$

So, we need to find

$$\nabla_{\vec{w}} L_i = \frac{\partial L_i}{\partial \hat{y}_i} \cdot \frac{\partial \hat{y}_i}{\partial z} \cdot \nabla_{\vec{w}} z$$

$$= \left(-\ln\left|\frac{y_i}{2} - 0.5 + \hat{y}_i\right|\right)'_{\hat{y}_i} \cdot \left(\frac{1}{1+e^{-z}}\right)'_{z} \cdot \nabla_{\vec{w}} z$$

Logistic Regression Neuron

- We need to find

$$\nabla_{\vec{w}} L_i = \frac{\partial L_i}{\partial \hat{y}_i} \cdot \frac{\partial \hat{y}_i}{\partial z} \cdot \nabla_{\vec{w}} z$$

$$= \left(-\ln\left|\frac{y_i}{2} - 0.5 + \hat{y}_i\right|\right)'_{\hat{y}_i} \cdot \left(\frac{1}{1+e^{-z}}\right)'_z \cdot \nabla_{\vec{w}} z$$

$$= -\frac{\text{sign}(y_i)}{\left|\frac{y_i}{2} - 0.5 + \hat{y}_i\right|} \cdot -\frac{1}{\left(1+e^{-\vec{w}\cdot\vec{x}_i}\right)^2} \cdot \left(-e^{-\vec{w}\cdot\vec{x}_i}\right) \cdot \nabla_{\vec{w}} \vec{w} \cdot \vec{x}_i$$

$$= -\frac{\text{sign}(y_i)}{\left|\frac{y_i}{2} - 0.5 + \hat{y}_i\right|} \cdot \hat{y}_i \frac{e^{-\vec{w}\cdot\vec{x}_i}}{\left(1+e^{-\vec{w}\cdot\vec{x}_i}\right)} \cdot \vec{x}_i$$

$$= -\frac{\text{sign}(y_i)}{\left|\frac{y_i}{2} - 0.5 + \hat{y}_i\right|} \cdot \hat{y}_i \cdot (1 - \hat{y}_i) \cdot \vec{x}_i$$

$$= \begin{cases} -(1-\hat{y}_i)\vec{x}_i & \text{if } y_i = 1 \\ \hat{y}_i \vec{x}_i & \text{if } y_i = -1 \end{cases}$$

$$= \begin{cases} -\frac{\vec{x}_i}{1+\exp(\vec{w}\cdot\vec{x}_i)} & \text{if } y_i = 1 \\ \frac{\vec{x}_i}{1+\exp(-\vec{w}\cdot\vec{x}_i)} & \text{if } y_i = -1 \end{cases} = -\frac{y_i \vec{x}_i}{1+\exp(y_i \times (\vec{w}\cdot\vec{x}_i))}$$

Logistic Regression Neuron - Updates

- We need found

$$\nabla_{\vec{w}} L_i == -\frac{y_i \vec{x}_i}{1+\exp(y_i \times (\vec{w} \cdot \vec{x}_i))}$$

So the update rule is going to be

$$\vec{w}_n = \vec{w}_o + \alpha \frac{y_i \vec{x}_i}{1+\exp(y_i \times (\vec{w} \cdot \vec{x}_i))}$$

If magnitude regularization is added then the rule becomes

$$\vec{w}_n = \vec{w}_o(1 - \alpha\lambda) + \alpha \frac{1}{1+\exp(y_i \times (\vec{w} \cdot \vec{x}_i))} \cdot y_i \vec{x}_i$$

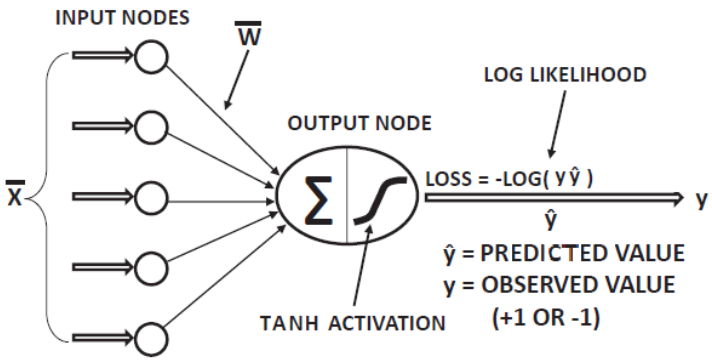
Notice that $p(error) = \begin{cases} 1 - \frac{1}{1+e^{-\vec{w} \cdot \vec{x}_i}} \text{ when } y_i = 1 \\ \frac{1}{1+e^{-\vec{w} \cdot \vec{x}_i}} \text{ when } y_1 = -1 \end{cases} = 1 - \frac{1}{1+e^{-y_i(\vec{w} \cdot \vec{x}_i)}}$

$$= \frac{1+e^{y_i(\vec{w} \cdot \vec{x}_i)} - e^{y_i(\vec{w} \cdot \vec{x}_i)}}{1+e^{y_i(\vec{w} \cdot \vec{x}_i)}} = \frac{1}{1+\exp(y_i \times (\vec{w} \cdot \vec{x}_i))} \text{ so}$$

$$\vec{w}_n = \vec{w}_o(1 - \alpha\lambda) + \alpha p(error) y_i \vec{x}_i$$

Hyperbolic Tangent Neuron

- Neuron is



Where

- cumulative loss is $\mathrm{L} = \sum_{i=1}^{n} L_i = -\ln \prod_{i=1}^{n} y_i \hat{y}_i$
- per example loss is $L_i = -\ln(y_i \hat{y}_i)$
- $\hat{y}$ = output of activation $\Phi(z) = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$
- activation input is $z = \vec{w} \cdot \vec{x}$

$$\text{So } \nabla_{\vec{w}} L_i = \frac{\partial L_i}{\partial \hat{y}_i} \cdot \frac{\partial \hat{y}_i}{\partial z} \cdot \nabla_{\vec{w}} z = (-\ln(y_i \hat{y}_i))'_{\hat{y}} \cdot \left(\frac{e^z - e^{-z}}{e^z + e^{-z}}\right)' \cdot \nabla_{\vec{w}} z$$

$$= -\frac{y_i}{y_i \hat{y}_i} \cdot \left(\frac{(e^z + e^{-z}) \cdot (e^z + e^{-z}) - (e^z - e^{-z})(e^z - e^{-z})}{(e^z + e^{-z})^2}\right)\Bigg|_{z = \vec{w}\vec{x}_i} \cdot \vec{x}_i$$

$$= -\frac{(1 - \hat{y}_i)}{\hat{y}_i} \cdot \vec{x}_i = \hat{x}_i - \frac{\vec{x}_i}{\hat{y}_i}$$

SVM Neuron as Perceptron Modification

- Alternative perceptron criterion is to minimize errors:

$$L_i^{per} = \max\{-y_i(\vec{w} \cdot \vec{x}_i), 0\}$$

  It is >0 whenever sign of $\vec{w} \cdot \vec{x}_i$ is not the same as sign of y_i, i.e. we made a mistake! so we update. Known as hinge loss.

- What if we want to minimize margin of error? To make it at least 1 we could have

$$L_i^{svm} = \max\{1 - y_i(\vec{w} \cdot \vec{x}_i), 0\}$$

  i.e. it is >0 whenever $1 > y_i(\vec{w} \cdot \vec{x}_i)$, so we update on error and when margin from separating hyperplane $\vec{w} \cdot \vec{x}_i$ is less than 1, i.e. we are "barely correct"!

  It is a shifted version of hinge-loss in perceptron.

- Of course, in computing gradient constant 1 is lost, so updates of SVM are *identical* to perceptron:

$$\vec{w}' = \vec{w}(1 - \alpha\lambda) + \begin{cases} \alpha y_i \vec{x}_i \text{ if } y_i \hat{y}_i < 0 \\ 0 \text{ otherwise} \end{cases}$$

Comparing Update rule for 'Linear' Neurons

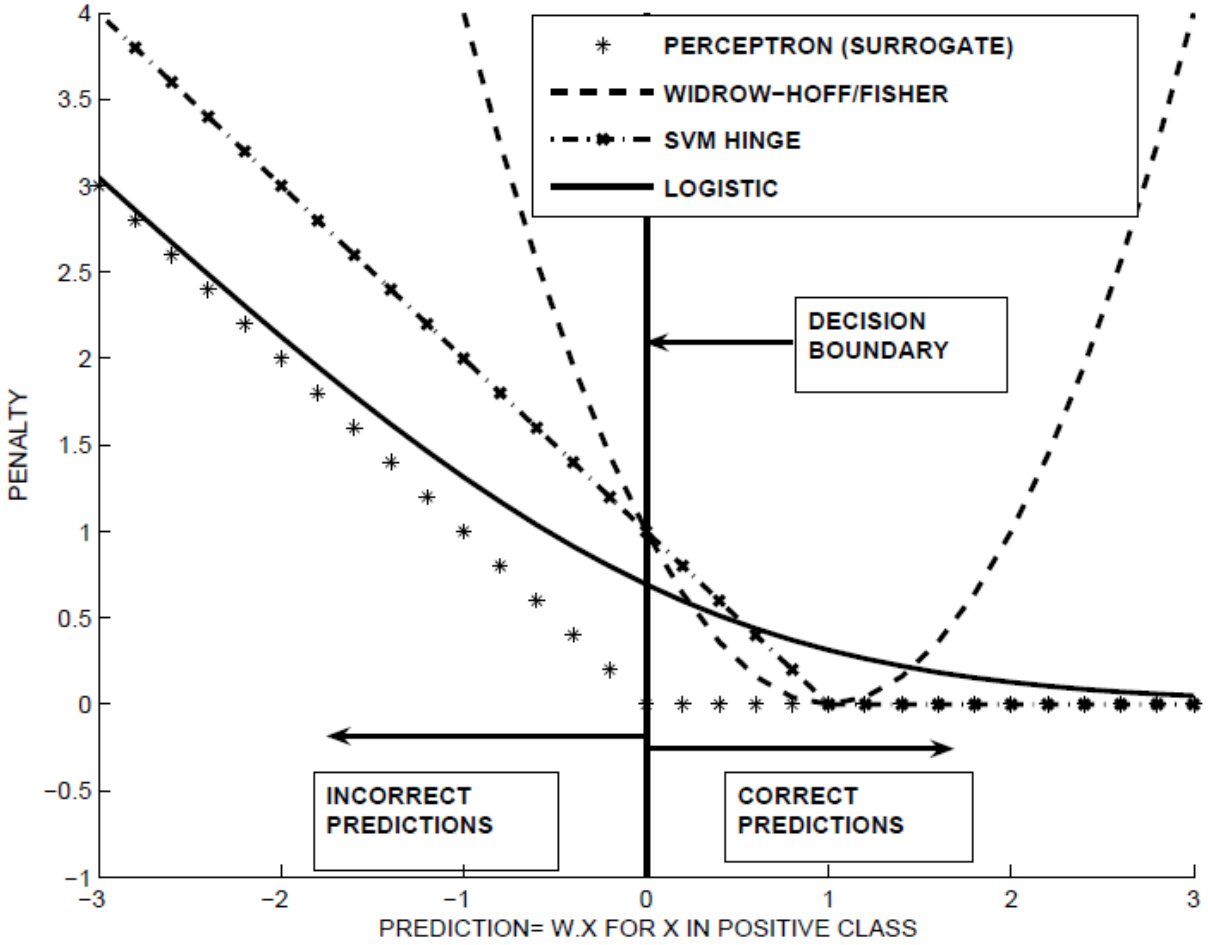
- The unregularized updates of the perceptron, SVM, Fisher LD, and logistic regression can be written in the form:

$$\vec{w}' = \vec{w} + \alpha\delta(\vec{x}_i, y_i)\vec{x}_i y_i$$

Where $\delta(\vec{x}_i, y_i)$ is a *mistake function*, which is:

- Raw mistake value $(1 - y_i(\vec{w} \cdot \vec{x}_i))$ for Fisher LD
- Mistake indicator, i.e. $\max\{0, -y_i(\vec{w} \cdot \vec{x}_i)\} > 0$ for perceptron.
- Margin/mistake indicator, i.e. whether $\max\{0,1 - y_i(\vec{w} \cdot \vec{x}_i)\} > 0$ for SVM.
- Probability of mistake for logistic regression

Comparing Loss for 'Linear' Neurons



Reading

Ch. 2.2.1-2.2.4